Finding a Long Directed Cycle

Harold N. Gabow [*] Shuxin Nie [*]

Abstract

For fixed k we present linear and almost-linear time algorithms to find a directed cycle of length $\geq k$, if one exists. We find a directed cycle of length $\geq \log n/\log\log n$ in polynomial time, if one exists. Under an appropriate complexity assumption it is known to be impossible to improve this by more than a $\log\log n$ factor. Our approach is based on depth-first search.

1 Introduction

For a given integer $k \geq 2$ we call a cycle in a directed or undirected graph *long* if it has $\geq k$ edges. Finding a long cycle is NP-hard if k is part of the input, since it includes the Hamiltonian cycle problem. A straightforward algorithm running in time $O(n^{k-1}m)$ shows the problem can be solved in polynomial time for fixed k. For undirected graphs the problem is known to be fixed-parameter tractable. The directed case is believed to be harder (see e.g., [4, p.19] and the complexity results summarized below). We know of no fixed-parameter tractability results for the directed case. This paper presents such algorithms, e.g., a time $O(m)$ algorithm. The approach is to exploit the structure of long cycles that is revealed by a depth-first search. Our techniques also lead to some improvements for finding long undirected paths and cycles.

Previous work Monien [10] appears to be the first to investigate fixed parameter algorithms for long paths and cycles. His basic algorithm works for both directed and undirected graphs. It uses a combinatorial idea of "representative subcollections." The algorithm finds a path of length (exactly) k, or determines no such path exists, in time $O(k!nm)$. Monien shows this algorithm can be used to find a longest undirected cycle in time $O((2\ell - 1)!nm)$, where ℓ is the longest cycle length.

Fellows and Langston investigated the problem of finding an undirected cycle of length $\geq k$, applying Robertson-Seymour theory [7] and also introducing the use of depth-first search [8]. The latter was extended by Bodlaender [4] who used dynamic programming to find a long undirected cycle in time $O(2^k k!n)$. This implies a longest undirected cycle can be found in time $O(2^\ell(\ell+1)!n)$, improving Monien's bound.

Alon et. al. [2] introduced the technique of "color coding" (random color assignments) to find paths, cycles and other small subgraphs. Their algorithm finds a directed path of length k, if one exists, in $2^{O(k)}m$ expected or $2^{O(k)}m\log n$ worst-case time. The factor m reduces to n for an undirected path. (Other bounds are possible too.) They find a cycle of length exactly k in a directed or undirected graph in $2^{O(k)}nm$ or $2^{O(k)}n^\omega$ expected time. Here ω is the exponent for matrix multiplication, $\omega < 2.376$. The derandomized versions of these algorithms have an extra $\log n$ factor in the running time. Improved bounds for finding a cycle of length exactly k are given in [3] (for directed and undirected graphs) and [13] (undirected graphs).

Björklund and Husfeldt [5] showed how to find an undirected path of length $\Omega((\log\ell/\log\log\ell)^2)$ in polynomial time, for ℓ the longest path length.

On the hardness side Björklund et. al. [6] prove two results concerning our problem: The first result assumes $P \neq NP$. In that case for any $\epsilon > 0$ no polynomial-time algorithm can find a cycle of length $\geq n^\epsilon$ in a given Hamiltonian directed graph. The second result assumes the Exponential Time Hypothesis of [9]. This is the stronger assumption that 3-SAT, on instances of n variables, has no subexponential ($2^{o(n)}$) time algorithm. In that case the above the lower bound of n^ϵ can be strengthened to $f(n)\log n$ for any function f that is $\omega(1)$, nondecreasing and computable in subexponential time. ([6] actually proves the lower bound $\log^{1+\epsilon} n$ for any $\epsilon > 0$; the same argument gives the above stronger assertion.)

Our contribution The algorithms of Monien and Alon et. al. find cycles of small length k efficiently. But the problem of finding a cycle of length $\geq k$ may entail finding a much longer, even Hamiltonian, cycle. We show how depth-first search resolves this difficulty by finding the longer cycles.

To be precise first consider undirected graphs. Recall that a long cycle has length $\geq k$. Theorem 4.1 shows that an undirected graph having a long cycle either has a long fundamental cycle in every dfs tree or has a long cycle of length $\leq 2k-4$. This enables us to find a long cycle using dfs plus the algorithm of [2]. The time is

[*]Department of Computer Science, University of Colorado at Boulder, Boulder, Colorado 80309-0430, USA. e-mail: {hal, Shuxin.Nie}@cs.colorado.edu

Copyright © 2004 by the Association for Computing Machinery, Inc. and the Society for industrial and Applied Mathematics. All Rights reserved. Printed in The United States of America. No part of this book may be reproduced, stored, or transmitted in any manner without the written permission of the publisher. For information, write to the Association for Computing Machinery, 1515 Broadway, New York, NY 10036 and the Society for Industrial and Applied Mathematics, 3600 University City Science Center, Philadelphia, PA 19104-2688

$2^{O(k)}n$ expected and $2^{O(k)}n\log n$ worst-case. For appropriate values of k and n this improves Bodlaender's bound of $O(2^k k!n)$ for a long undirected cycle.

The directed case is analogous. We define a "lowpoint cycle" based on the lowpoint values of Tarjan's strong connectivity algorithm [11]. Theorem 2.1 shows that a strongly connected directed graph having a long cycle either has a long lowpoint cycle in every dfs tree or has a long cycle of length $\le k^3$. This allows us to find a long directed cycle in these time bounds (Theorem 3.1): For bounds essentially linear in m we have expected time $2^{O(k^2)}m$ and worst-case time $2^{O(k^2)}m\log n$ or $O(k^2!m)$. For bounds with better dependence on k we have expected time $k^{2k}2^{O(k)}nm$ and worst-case time $k^{2k}2^{O(k)}nm\log n$ or $O(k^{3k}nm)$. The last two bounds allow us to find a directed cycle of $\ge \log n/\log\log n$ edges in polynomial time (Corollary 3.1). The hardness results of Björklund et. al. cited above show this is close to the best one can hope for (e.g. under the Exponential Time Hypothesis it is within a $\log\log n$ factor of best possible).

Returning to our theorem on lowpoint cycles, Fig.1 gives a simple example showing the k^3 upper bound cannot be reduced below $(k-1)(k-2)$. A more involved argument proves the theorem with this $(k-1)(k-2)$ upper bound. So this version of the lowpoint cycle theorem is existentially tight. Similarly our undirected long cycle theorem is tight (Fig. 3).

Finally returning to undirected graphs, we improve the algorithm of Björklund and Husfeldt cited above by a $\log\log n$ factor: An undirected cycle of length $\Omega(\log^2\ell/\log\log\ell)$ can be found in polynomial time, for ℓ the longest path length.

Organization of the paper §2 presents our results relating depth-first search and long directed cycles. §3 gives the algorithms based on these results. §4 discusses undirected graphs. Appendix A proves the tight version of the lowpoint cycle theorem.

Terminology A *walk* can have repeated vertices. A *vw-walk* goes from vertex v to vertex w. A *closed walk* has its first and last vertices identical. A *path* is a simple walk (it has no repeated vertices). A *cycle* is a simple closed walk.

We represent a walk by listing its vertices, e.g., v, w, x. We also allow walks to be included in the list, e.g., if (v, w) is an edge and X is a walk beginning at w then v, X denotes a walk that is 1 edge longer than X. Sometimes we write v, w, X for the same walk to remind the reader of the first edge (v, w). This will not cause any confusion.

A *shortcut* of a walk W is a walk obtained by deleting 0 or more internal vertices of W. So if W is the walk $v(0), v(1), \ldots, v(\ell)$ a shortcut is a walk $v(i_0), v(i_1), \ldots, v(i_r)$, where $0 = i_0 < i_1 < \ldots < i_r = \ell$ and $(v(i_j), v(i_{j+1}))$ is an edge for $0 \le j < r$.

If P is a walk we use P to denote a set of vertices or edges, as is convenient; the exact meaning will be clear from context. However the notation $|P|$ always denotes the number of edges in P.

2 Depth-first Search

This section explores the structure imposed on long cycles by depth-first search. Our approach to finding a long cycle is to first look for a long cycle that is easily found by dfs. If the dfs fails to find a long cycle the graph has several useful properties established in this section. For undirected graphs the well-known principle is that the graph of a failed dfs is sparse, i.e., it contains $O(kn)$ edges. Although this is not true in digraphs a weaker property holds: The vertices joined by an edge are close to each other in terms of depth (Lemma 2.4). A second property is that if a long cycle actually exists (and has not been found by the dfs) a long cycle of length $O(k^2)$ exists (Theorem 2.1). We also prove an analog of the latter result for undirected graphs.

We begin by fixing some notation. Throughout this section we assume the given graph G is directed. We also assume G is strongly connected (since any cycle lies within one strongly connected component). We fix a dfs tree T. r denotes the root of T. $d(v)$ is the depth of a vertex v in T. $nca(v, w)$ is the nearest common ancestor of vertices v and w in T. T_v is the subtree of T rooted at v. As usual assume each vertex is identified by its dfs discovery number (i.e., its preorder number in T). So for instance the "lowest vertex" in a set S refers to the vertex of S with lowest discovery number.

If P is a path or cycle containing vertices v and w in that order then $P[v, w]$ denotes the subpath of P from v to w. For the dfs tree T and v an ancestor of w in T, $T[v, w]$ denotes the path of tree edges from v to w. We extend this notation to allow open-sided intervals, e.g., $P(v, w]$ is the path $P[v, w] - v$.

We start with some basic properties of dfs proved by Tarjan [11, 12].

LEMMA 2.1. *(i) Any vw-path with $v < w$ contains a common ancestor of v and w.*

(ii) Any cycle C consists of descendants of its lowest vertex.

(iii) Any vw-path with w a proper ancestor of v contains a back edge directed to an ancestor of w.

Proof. (i) is proved in [12]. For (ii) let v be the lowest numbered vertex of C. For each $w \in C$ apply (i) to the path $C[v, w]$. It shows v is an ancestor of w. For (iii) add the forward edge (w, v) to change the path into a cycle C. Apply part (ii) to C. The edge entering the lowest vertex of C proves (iii).

This consequence will be useful.

LEMMA 2.2. *For any vertex b, any vw-path contains a back edge directed to a vertex of $T[r,b)$ if (i) or (ii) holds:*

(i) $v \in T_b$ and either $w \in T[r,b)$ or $w > \max\{x : x \in T_b\}$.

(ii) $v \notin T[r,b)$, $v < b$ and $w \in T_b$.

Proof. (i) Assume $v \in T_b$. If $w \in T[r,b)$ the lemma is Lemma 2.1(iii). Suppose $w > \max\{x : x \in T_b\}$. Lemma 2.1$(i)$ shows the path contains a common ancestor c of v and w. Since $v \in T_b$ and $w \notin T_b$, c is a proper ancestor of b. Now apply Lemma 2.1(iii) as before.

(ii) The argument is similar. We have $v < b \le w$, so Lemma 2.1(i) shows the path contains a common ancestor c of v and w. Since $v < b$ we have $v \notin T_b$ and $w \in T_b$. Thus c is a proper ancestor of b. Since $v \notin T[r,b)$ we have $c \neq v$. Now apply Lemma 2.1(iii) as before.

We use the lowpoint values that are the basis of Tarjan's strong connectivity algorithm [11]. Recall the definition from [11, 1], which we simplify using the assumption that G is strongly connected: Each vertex $v \neq r$ has a value *lowpoint*(v). If $y =$ *lowpoint*(v) then there is a descendant x of v with a back or cross edge (x,y) where $y < v$ and y is as low as possible. The existence of *lowpoint*(v) is guaranteed by strong connectivity.

The *lowpoint path* for v, denoted L_v, is a path from v to the root r defined as follows: L_r is the length 0 path consisting of vertex r. If $v \neq r$ and *lowpoint*$(v) = y$ with corresponding edge (x,y) then $L_v = T[v,x], y, L_y$. (We may have $v = x$. If more than one vertex qualifies as x the choice of x is arbitrary.)

LEMMA 2.3. *For any vertex v, L_v is simple.*

Proof. All vertices on $T[v,x]$ have the same lowpoint value y. So as we traverse L_v, *lowpoint* values never increase. Hence no vertex of $T[v,x]$ belongs to L_y. An easy induction shows L_v is simple.

Let (v,w) be a nontree edge. Let a be the first vertex in L_w that is an ancestor of v. Define the *lowpoint cycle* of edge (v,w) by

$$C_{vw} = v, w, L_w[w,a], T[a,v].$$

For example a back edge (v,w) has $C_{vw} = v, w, T[w,v]$. It is easy to check that C_{vw} is simple. Hence C_{vw} is a long cycle if it has $\geq k$ edges.

This section studies the properties of graphs having no long lowpoint cycles. The first property, as already mentioned, is that the endpoints of an edge have similar depths:

LEMMA 2.4. *If no lowpoint cycle is long then the following inequalities hold:*

(i) For every back edge (v,w), $d(v) \le d(w) + k - 2$.

(ii) For every cross edge (v,w), $d(v) - k + 4 \le d(w) \le d(v) + (k-3)(k-2) - 1$.

(iii) For every forward edge (v,w), $d(w) \le d(v) + (k-2)^2$.

(iv) For every edge (v,w), $d(v) \le d(w) + k - 2$.

Proof. (i) If (v,w) is a back edge then $C_{vw} = v, w, T[w,v]$ is long when $|T[w,v]| \ge k-1$, i.e., when $d(v) \ge d(w) + k - 1$.

(ii) For any cross edge (v,w) vertex a is a common ancestor of v and w (Lemma 2.1(i) applied to path $L_w[w,a], T[a,v]$). Hence $a \neq v, w$ and both subpaths $L_w[w,a]$ and $T[a,v]$ of C_{vw} have ≥ 1 edge. So C_{vw} is long if either subpath has $\ge k-2$ edges.

To prove the first inequality of (ii) suppose $d(w) < d(v) - k + 4$. Then $d(a) \le d(w) - 1 \le d(v) - k + 2$. Hence $|T[a,v]| \ge k-2$ and C_{vw} is long. We conclude $d(v) - k + 4 \le d(w)$, i.e., the first inequality holds.

Notice that (iv) now follows from (i) and the first inequality of (ii) (since a tree or forward edge (v,w) has $d(w) > d(v)$).

To prove the second inequality of (ii) suppose $d(w) > d(v) + (k-3)(k-2) - 1$. Since $d(a) \le d(v) - 1$, (iv) shows $|L_w[w,a]| \ge (d(w) - d(a))/(k-2) > (k-3)(k-2)/(k-2) = k-3$. Thus again C_{vw} is long and the second inequality holds.

(iii) Let (v,w) be a forward edge with $d(w) > d(v) + (k-2)^2$. Since a is an ancestor of v, (iv) implies $|L_w[w,a]| \ge (d(w) - d(v))/(k-2) > k-2$. Thus $|L_w[w,a]| \ge k-1$ and C_{vw} is long.

For $k \le 3$ lowpoint cycles have this strong property: The existence of a long cycle implies the existence of a long lowpoint cycle. This follows easily from Lemma 2.4. (For $k = 2$ any (lowpoint) cycle is long. For $k = 3$ if no lowpoint cycle is long then there are no cross or forward edges, and every back edge goes from child to parent. Hence there are no long cycles at all.) For $k > 3$ simple examples show this strong property does not hold (see also Fig.1). Instead we have this second property:

THEOREM 2.1. *In a strongly connected digraph having a long cycle, either every dfs tree has a long lowpoint cycle or some long cycle has at most k^3 edges.*

As an example Fig.1 shows that the k^3 upper bound in the theorem cannot be reduced below $\Theta(k^2)$. We

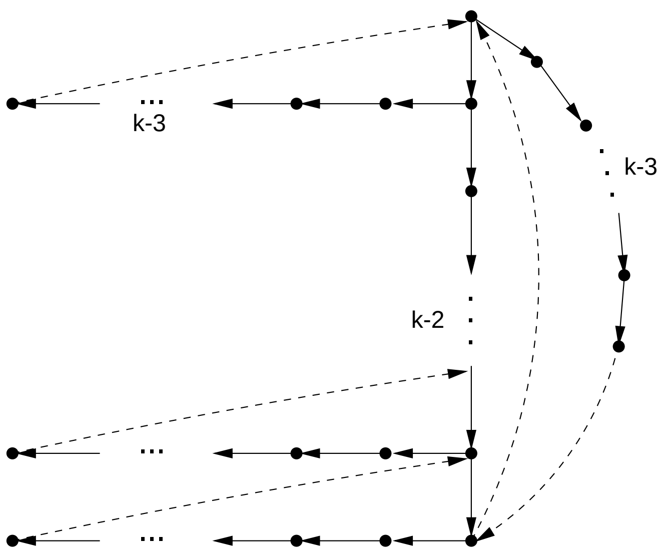

Figure 1: Lower bound graph for Theorem 2.1, when $k \geq 4$. All k lowpoint cycles have length exactly $k-1$. The unique long cycle is Hamiltonian and has length $(k-3)+1+(k-2)^2 = (k-1)(k-2)$.

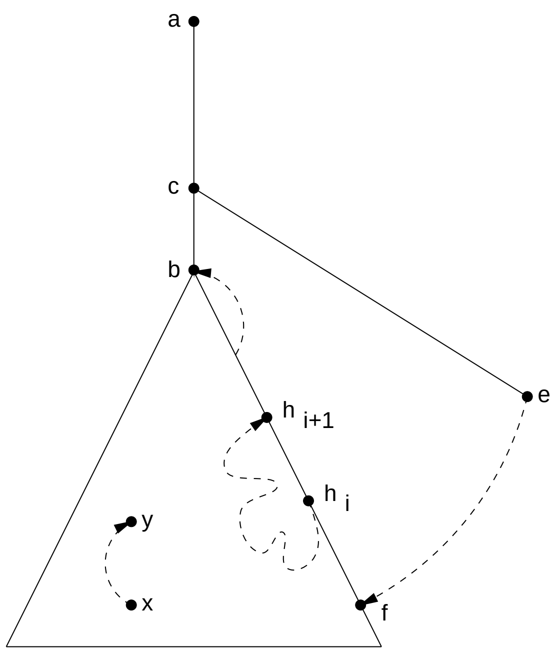

Figure 2: Analysis of the long cycle C for Theorem 2.1.

indicate at the end of this section how to actually improve the k^3 upper bound. However the k^3 version of the theorem is adequate for all the applications in this paper.

The theorem is proved in Lemmas 2.5–2.9. In fact an upper bound of $2k-4$ on the long cycle length is proved in all the lemmas except the last one, where the k^3 bound appears. Since the ultimate goal is to prove the tight version of the theorem we make some estimates to greater precision than needed for just the k^3 bound.

Fix some dfs tree that has no long lowpoint cycle. Let C be a long cycle containing as few nontree edges as possible. Lemmas 2.5–2.9 show that $|C| \leq k^3$. Let a be the first vertex of C and assume C consists of descendants of a (Lemma 2.1).

In the next lemma $C[w,a]$ and $T(a,w)$ denote vertex sets.

Lemma 2.5. *Let (v,w) be the first cross edge in C, if it exists. Suppose $C[a,v]$ has no back edge and $C[w,a] \cap T(a,w) = \emptyset$. Then $|C| \leq 2k-4$.*

Proof. The first hypothesis implies $C[a,v]$ consists of tree and forward edges. Hence $C[a,w]$ is a shortcut of the path $T[a,v], w$. Take $b = nca(v,w)$, so

$$C \text{ is a shortcut of } T[a,b], T[b,v], w, C[w,a].$$

Write $D = T[a,w], C[w,a]$. The second hypothesis implies D is a simple cycle. Since D has one less nontree edge than C, D is not long, i.e., $|D| < k$. Write

$$D = T[a,b], T[b,w], C[w,a].$$

Since $b \neq w$ the two displayed relations give

$$|C| \leq (|D|-1) + (|T[b,v]|+1) = |D| + |T[b,v]|.$$

Since the lowpoint cycle C_{vw} is not long, and it includes $T[b,v], w$ plus at least one edge of L_w, we get $|T[b,v]| \leq k-3$. In summary $|C| \leq (k-1)+(k-3) = 2k-4$.

Write

$$(x,y) = \text{the first back edge in } C.$$

Fig.2 shows a, x, y and additional notation to be used in the proofs. Some of the notation does not get defined until Lemma 2.9 but we point to the figure as new items get introduced.

We first treat the case $y = a$.

Lemma 2.6. *If $y = a$ then $|C| \leq 2k-4$.*

Proof. C must contain a cross edge, since otherwise $C[a,x]$ consists of tree and forward edges, making C a shortcut of the lowpoint cycle C_{xy}. So let (v,w) be the first cross edge in C. The hypotheses of Lemma 2.5 are satisfied: $C[a,v]$ has no back edge by the lemma's hypothesis. $C[w,a] \cap T(a,w) = \emptyset$ by Lemma 2.1(iii). Now Lemma 2.5 gives the desired conclusion.

We turn to the case $y \neq a$. Write

$$\begin{aligned} b &= \text{the shallowest vertex in } T(a,y] \\ &\quad \text{that is the head of a back edge of } C; \\ f &= \text{the first vertex of } C \text{ in } T_b. \end{aligned}$$

See Fig.2. Note that vertex b exists since $y \neq a$.

The next lemma shows how C behaves with respect to b. In all parts of the lemma paths are interpreted as vertex sets.

LEMMA 2.7. *(i)* $C[f,a] \cap T(a,b) = \emptyset$.
(ii) $C[a,f) \subseteq T[a,b) \cup \{v : v \geq b + |T_b|\}$.
(iii) $C[f,a] \subseteq \{v : v < b + |T_b|\}$.

Proof. (i) The definitions of a and b imply no back edge of $C[f,a)$ is directed to a vertex of $T[r,b)$. Now apply Lemma 2.2(i) with $v = f$, w any vertex of $C(f,a)$, and $b = b$.

(iii) is similar. (Note that we trivially have $a < b \leq b + |T_b|$.)

(ii) The definition of f implies $C[a,f]$ has no back edge. Now apply Lemma 2.2(ii) with v any vertex of $C[a,f)$, $w = f$ and $b = b$. Note that $v < b + |T_b|$ implies $v < b$, again by the definition of f.

The next two lemmas complete the proof of Theorem 2.1 by handling the case $y \neq a$.

LEMMA 2.8. *If* $y \neq a$ *and* $C[a,f)$ *contains a cross edge then* $|C| \leq 2k - 4$.

Proof. Let (v,w) be the first cross edge in C. We will prove the hypotheses of Lemma 2.5. This immediately implies the lemma. The definition of f implies $C[a,v]$ has no back edge. So we need only show $C[w,a] \cap T(a,w) = \emptyset$.

Suppose this is false. Choose w' as the first proper ancestor of w in $C[w,a)$. We will show the following:

(i) w is not an ancestor or descendant of b.

(ii) w' occurs after f in C.

Before establishing claims (i)–(ii) let us show they imply the desired contradiction: Since $w \in C[a,f)$, Lemma 2.7(ii) and claim (i) imply $w \geq b + |T_b|$. Claim (ii) and Lemma 2.7(iii) imply $w' < b + |T_b|$. Using again claim (i) shows $w' < b$. Thus since w descends from w' and $w' < b < w$, b descends from w'. But Lemma 2.7(i) and claim (ii) show w' is not an ancestor of b, a contradiction.

To show (i), first note that $w \in C[a,f)$ ensures w does not descend from b. Lemma 2.1(iii) shows $C[w,w']$ enters w' by a back edge. Clearly this shows w is not an ancestor of b. Thus (i) holds. We also get (ii) by the definition of f.

LEMMA 2.9. *If* $y \neq a$ *and* $C[a,f)$ *does not contain a cross edge then* $|C| \leq k^3$.

Proof. Write $C = C[a,f], C[f,a]$. We will analyze the length of the two pieces. Let C enter f on the edge (e,f). Fig.2 illustrates this and the additional notation introduced in this argument. Since (e,f) enters T_b, (e,f) is a forward or cross edge. Let $c = nca(b,e)$. Clearly $c \neq b$. (We can have $c = e$ if (e,f) is a forward edge.)

We first estimate $|C[f,a]|$. Define a sequence $h_0 = f,\ h_1, \ldots,\ h_j = a$ of ancestors of f that are "high-water marks" of C. More precisely $h_0 = f$ and in general

$$h_{i+1} = \text{ the ancestor of } h_i \text{ that occurs first in } C(h_i, a].$$

Clearly a eventually occurs as some h_j. Lemma 2.7(i) shows $b = h_{j-1}$.

Consider any segment $C[h_i, h_{i+1}]$, $0 \leq i < j$. The definition implies $C[h_i, h_{i+1}], T[h_{i+1}, h_i]$ is a (simple) cycle. It avoids (e,f) and so has fewer nontree edges than C. Thus it is not long, $|C[h_i,h_{i+1}]| + |T[h_{i+1},h_i]| \leq k-1$. Adding together these j inequalities and noting that $j \leq |T[b,f]| + 1$ gives

$$|C[f,a]| + |T[a,f]| \leq (|T[b,f]| + 1)(k-1).$$

The lemma's hypothesis implies $C[a,e]$ consists of tree and forward edges. Thus

$$\begin{aligned} |C[a,f]| &= 1 + |C[a,e]| \leq 1 + |T[a,e]| \\ &= 1 + |T[a,c]| + |T[c,e]| \leq |T[a,b]| + |T[c,e]|. \end{aligned}$$

Adding the 2 displayed inequalities and noting that $|T[a,f]| = |T[a,b]| + |T[b,f]|$ gives

$$(2.1) \quad |C| \leq (k-1) + |T[c,e]| + |T[b,f]|(k-2).$$

Suppose (e,f) is a forward edge. Lemma 2.4(iii) shows $|T[b,f]| \leq |T[e,f]| - 1 \leq (k-2)^2 - 1$. Also $c = e$. Now (2.1) becomes $|C| \leq (k-1) + ((k-2)^2 - 1)(k-2) = (k-2)^3 + 1$.

Suppose (e,f) is a cross edge. Note that $c = nca(b,e) = nca(f,e)$. Hence $|T[c,e]| \leq |C_{ef}| - 2 \leq k-3$. Observe that C_{ef} consists of edge (e,f) plus a path P from f to an ancestor of c, plus at least one tree edge. Since C_{ef} is not long, $|P| \leq k-3$. Lemma 2.4(iv) shows $|T[b,f]| < d(c) - d(b) \leq |P|(k-2) \leq (k-3)(k-2)$. Now (2.1) becomes $|C| \leq (k-1) + (k-3) + (k-3)(k-2)^2 = 2(k-2) + (k-3)(k-2)^2 \leq k^3$.

This establishes Theorem 2.1. A more careful analysis reduces the k^3 term to $(k-1)(k-2)$. The example of Fig.1 shows this bound is existentially tight for $k \geq 4$. (It is tight for $k < 4$ as well.) The idea of the tight analysis is to use the lowpoint cycle C_{ef} to shortcut $C[f,a]$ from $\Theta(k^3)$ edges to $\Theta(k^2)$. The tight analysis is given in the Appendix. (The analysis of the next section uses the better bound k^2. However the reader will notice that the results hold, with a few trivial changes, even if we use the weaker bound k^3.)

3 Algorithms

The bulk of this section is devoted to several algorithms that establish the following:

THEOREM 3.1. *A long cycle in a directed graph can be found, if one exists, in the following time bounds:*
(i) expected time $2^{O(k^2)}m$;
(ii) worst-case time $2^{O(k^2)}m\log n$ or $O(k^2!m)$;
(iii) expected time $k^{2k}2^{O(k)}nm$;
(iv) worst-case time $k^{2k}2^{O(k)}nm\log n$ or $O(k^{3k}nm)$.

We rely on two subroutines for finding long paths. Monien's algorithm [10] starts with a given $v \in V$ and finds a length k vw-path for every $w \in V$ for which such a path exists. The time is $O(k!m)$. We refer to this as the *representative algorithm*.

The algorithm of Alon et. al. [2] has both randomized and deterministic versions. The randomized version starts with a given $v \in V$ and searches for a length k vw-path for every $w \in V$. Consider any $w \in V$ having such a path. The algorithm finds a length k vw-path in expected time $2^{O(k)}m$. More precisely one execution of the algorithm runs in time $O(k2^km)$ and finds a length k vw-path with probability $> e^{-k}$. The derandomized version of this algorithm solves the same problem as Monien: given some $v \in V$ it finds a length k vw-path for every $w \in V$ for which such a path exists. The worst-case time is $2^{O(k)}m\log n$. We refer to these two algorithms of [2] as the *color coding algorithms.*

All our algorithms to find a long cycle are implementations of the following procedure, which is a straightforward application of Theorem 2.1:

Step 1. Search for a long lowpoint cycle. If one is found return it, else continue.
Step 2. For $\ell = k, k+1, \ldots, (k-1)(k-2)$, search for a cycle of length ℓ. If one is found return it. If none is found declare there are no cycles of length $\geq k$.

Step 1 is implemented in time $O(km)$ as follows. First compute all *lowpoint* values. Also label the vertices so we can test if a given vertex descends from another in $O(1)$ time [11, 1]. This can all be done in $O(m)$ time. With this information it is easy to compute a lowpoint cycle C_{vw} in time proportional to its length. Step 1 computes lowpoint cycles for nontree edges, until either a long lowpoint cycle is found or all nontree edges are exhausted. Clearly Step 1 takes total time $O(km)$.

For Step 2 we first present two simple implementations that achieve $O(((k-1)(k-2))!nm)$ worst-case and $2^{O(k^2)}nm$ expected time respectively. Both implementations search for a cycle of length k by searching for a vw-path of length $k-1$ with $wv \in E$. Suppose we use the representative algorithm to search for the path. Each iteration of Step 2 takes time $O((\ell-1)!nm)$. This gives total time equal to the above worst-case bound. Next suppose we use randomized color coding. A single execution of Step 2 calls the color coding algorithm once for each start vertex v for each value of ℓ. If a long cycle exists, the expected number of repetitions of Step 2 to find a long cycle is $< e^{k^2}$. This gives total expected time equal to the above expected time bound.

We now improve these two time bounds to achieve parts (i) and (ii) of Theorem 3.1, which are linear (or close to linear) in m. For each vertex v let $G_v(h)$ denote the subgraph induced by the vertices of depth $\leq h$ in T_v (i.e., the vertex set is $\{w : w \in T_v,\ d(w) \leq d(v) + h\}$).

LEMMA 3.1. *Suppose no lowpoint cycle is long. Let C be a cycle with lowest vertex a.*
(i) C is contained in $G_a(k|C|)$.
(ii) Any vertex $w \in C$ is contained in $G_a(k^2|C[a,w]|)$.
(iii) For any integer h, the total number of edges contained in all subgraphs $G_v(h)$, $v \in V$ is $O(hm)$.

Proof. (i) Lemma 2.4(iv) implies that if $d(w) > d(a) + (k-2)|C|$ then any wa-path has length $> |C|$.

(ii) Similarly Lemma 2.4(ii)–(iii) implies that for $w \in C$, $d(w) \leq d(a) + (k-2)^2|C[a,w]|$.

(iii) Any given vertex w belongs to $\leq h+1$ subgraphs $G_v(h)$, since v must be one of the first $h+1$ ancestors of w.

To find a cycle of length ℓ in Step 2, for every $a \in V$ we search $G_a(k\ell)$ for a cycle beginning at vertex a. This is correct by Lemma 3.1(i). Lemma 3.1(iii) shows the total number of edges searched is $O(k\ell m)$. Using the representative algorithm one iteration of Step 2 takes time $O(\ell!k\ell m)$. This gives total time $O(((k-1)(k-2))!k^3m) = O(k^2!m)$, achieving the second bound of Theorem 3.1(ii). The other bounds of (i)–(ii) follow similarly.

To achieve the remaining bounds (iii)–(iv) we first present an algorithm to find a cycle of length between k and h inclusive, for given integers $k \leq h$. It is convenient to present a randomized algorithm and then derandomize it.

Suppose a cycle of the desired length exists, say cycle C. Consider 2 vertices $a, b \in C$ such that

$|C[a,b]| = k$. The algorithm works by guessing a and b and then attempting to construct the desired cycle from a length k ab-path and a length $\le h-k$ ba-path. In the following algorithm, for $Q \subseteq V$, Q^- denotes the set $Q - \{a,b\}$.

```
choose 2 random vertices $a, b \in V$
$S \leftarrow \emptyset$
**while** $|S| < k$ {
    $Q \leftarrow$ a $ba$-path of length $\le h-k$ in $G-S$
    **if** $Q$ does not exist **then return** "fail"
    $P \leftarrow$ an $ab$-path of length $k$ in $G-Q^-$
    **if** $P$ exists **then return** cycle $P, Q$
    choose a random vertex $v \in Q^-$
    add $v$ to $S$ }
```

We will analyze this algorithm and show it leads to the following result.

LEMMA 3.2. *A directed cycle of length between k and h can be found, if one exists, in expected time $n^2h^k2^{O(k)}m$ and worst-case time $n^2h^k2^{O(k)}m \log n$ or $O(n^2h^kk!m)$.*

Proof. We first show that if a cycle C of length between k and h exists, the algorithm finds such a cycle with probability $\ge 1/(n^2h^{k-1})$. The probability we guess a and b correctly is $\ge 1/n^2$. Suppose we have guessed a and b correctly, and furthermore for some $i \ge 0$, the first i iterations of the **while** loop have each guessed a vertex $v \in C(a,b)$. A path Q of the desired type exists, since $C[b,a]$ will do. Suppose P does not exist. Clearly this implies $Q^- \cap C(a,b) \ne \emptyset$. We choose a $v \in Q^- \cap C(a,b)$ with probability $\ge 1/(h-k)$. We conclude that the first $k-1$ iterations of the **while** loop guess the $k-1$ vertices of $C(a,b)$ with probability $\ge 1/(h-k)^{k-1}$, unless one of them returns a desired cycle. If the kth iteration starts with $S = C(a,b)$ then it is guaranteed to find a desired cycle, since $C[b,a]$ guarantees the existence of Q and $C[a,b]$ guarantees the existence of P. This gives the desired success probability for the algorithm.

It is trivial to derandomize this algorithm: The probability space has $O(n^2h^{k-1})$ points, so we can run the algorithm on each point. The worst-case number of runs to find a desired cycle or show that none exists is $O(n^2h^{k-1})$. From now on we will deal only with the deterministic version of the algorithm.

We turn to the implementation and time for the algorithm. For each of $O(n^2h^{k-1})$ points there are at most k iterations of the **while** loop. Q is found by breadth-first search in time $O(m)$. Suppose P is found using the representative algorithm. The total time is $O(n^2h^{k-1}kk!m) = O(n^2h^kk!m)$.

Next suppose P is found using randomized color coding. A single execution of our overall algorithm works by visiting each point of the probability space, and executing the randomized color coding algorithm (at most) once in each of k iterations. So a single execution uses (worst-case) time $n^2h^{k-1}k2^{O(k)}m$. It finds a desired cycle with probability $> e^{-k}$ if one exists. This gives a randomized algorithm that finds a desired cycle in expected time $n^2h^k2^{O(k)}m$ if one exists.

We implement Step 2 using this algorithm, with $h = (k-1)(k-2)$. The running time can be improved by restricting the choice of b, as follows. We can assume that a is the lowest vertex in the cycle to be found. Lemma 3.1(ii) shows b is in $G_a(k^3)$. Incorporating this restriction on b into the algorithm, a given vertex will be chosen as b $O(k^3)$ times. So there are $O(nk^3)$ choices of the pair a,b. The randomized color coding version of our algorithm has expected time $nk^3k^{2k}2^{O(k)}m = k^{2k}2^{O(k)}nm$. This gives part ($iii$) of Theorem 3.1, and the first bound of part (iv) is immediate. For the second bound the representative version of our algorithm has worst-case time $O(nk^3k^{2k}k!m) = O(k^{3k}nm)$. This completes the proof of Theorem 3.1.

Taking $k = \log n/\log\log n$ in Theorem 3.1(iv) gives the following:

COROLLARY 3.1. *A directed cycle that is either a longest cycle or has length $\ge \log n/\log\log n$ can be found in polynomial time.*

4 Undirected Graphs

This section shows how the depth-first search approach gives some improvements in algorithms for finding long undirected paths and cycles. We begin with an analog of Theorem 2.1.

THEOREM 4.1. *In a connected undirected graph having a long cycle, either every dfs tree has a long fundamental cycle or some long cycle has at most $2k-4$ edges.*

Before presenting the proof note that the example of Fig.3 shows the bound of the theorem is tight. Also, Monien [10] proves a closely related fact (that is the basis of his algorithm to find a longest cycle): In an undirected graph with diameter d, if $k \ge d$ then the existence of a long cycle implies the existence of a long cycle of $\le 2k+1$ edges. A slight advantage of Theorem 4.1 is that it avoids calculation of the diameter (see Theorem 4.2). Finally note that the graph of Fig.1 has diameter $\Theta(k)$ and unique long cycle length $\Theta(k^2)$. Hence if there is a directed analog of Monien's result it has a bound of $\Omega(k^2)$.

Proof. The lemma is clear if $k = 2$ (multigraphs are allowed) so assume $k \ge 3$. Fix a dfs tree that has no long

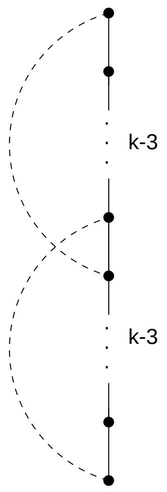

Figure 3: Lower bound graph for Theorem 4.1, when $k \geq 4$. The 2 fundamental cycles have length $k-1$; the unique long cycle is Hamiltonian and has length $2k-4$.

fundamental cycle. Let C be a long cycle containing as few back edges as possible. We will show $|C| \leq 2k-4$. Let a be the first vertex of C and assume C consists of descendants of a. (Such an a exists by the undirected version of Lemma 2.1(i).)

At most one edge of C passing through a is a tree edge. In proof, suppose on the contrary that C contains edges from a to two children b, b'. The undirected version of Lemma 2.1(i) implies $C[b,b']$ contains an ancestor of b and b'. Clearly this ancestor is a, so $a \in C[b,b']$. But the same argument shows $a \in C[b',b]$, contradicting the simplicity of C.

We conclude that C contains a back edge incident to a, say (f,a). Orient C to make f the second vertex. As in the proof of Lemma 2.9 define a sequence high-water marks $h_0 = f,\ h_1, \ldots,\ h_j = a$; more precisely $h_0 = f$ and in general

$$h_{i+1} = \text{ the ancestor of } h_i \text{ that occurs first in } C(h_i, a].$$

Clearly a eventually occurs as some h_j.

We claim that any segment $C[h_i, h_{i+1}]$, $0 \leq i < j$ has length $\leq k-2$. When the segment is a single tree edge (from child h_i to parent h_{i+1}) this follows from $k \geq 3$. In the opposite case, the definition of the high-water mark implies $C[h_i, h_{i+1}], T[h_{i+1}, h_i]$ is a cycle. It avoids (f,a) and so has fewer back edges than C. Thus it is not long, i.e., $|C[h_i, h_{i+1}]| + |T[h_{i+1}, h_i]| \leq k-1$, and the desired inequality follows.

Take index i as the smallest index with $|C[h_0, h_i]| > k-2$. The claim shows $i > 1$. Consider the cycle

$$D = C[h_0, h_{i-1}], T[h_{i-1}, a], h_0 = f.$$

Since the fundamental cycle of (h_0, a) is not long, and $i > 1$, we have $|T[h_{i-1}, a], h_0| \leq (k-1) - 1 = k-2$. Now the choice of i implies $|D| \leq (k-2) + (k-2) = 2k-4$.

Assume $|D| < k$, since otherwise D is long and we are done. Now consider the cycle

$$D' = C[h_0, h_i], T[h_i, a], h_0.$$

The choice of i guarantees D' is long (even if $h_i = a$). Replacing the subpath $C[h_{i-1}, h_i]$ of D' by $T[h_{i-1}, h_i]$ gives D. Hence $|D'| \leq |D| + |C[h_{i-1}, h_i]| - |T[h_{i-1}, h_i]| \leq (k-1) + (k-2) - 1 = 2k-4$, where we bound $|C[h_{i-1}, h_i]|$ using the claim.

Our algorithm to find a long cycle in an undirected graph is a simple application of the theorem: We begin by doing a dfs, stopping as soon as some back edge has a long fundamental cycle. If no such cycle is found we then search for a long cycle of length $\leq 2k-4$.

Suppose the dfs terminates unsuccessfully. This implies each vertex has $< k$ back edges so $m = O(kn)$. We search for long cycles of length $\leq 2k-4$ using the graphs $G_v(\cdot)$ defined in §3. Specifically to find a cycle of length ℓ, for every $a \in V$ search $G_a(k\ell)$ for a cycle beginning at vertex a. This is correct by the undirected analog of Lemma 3.1(i) (since each vertex of C is within $\lfloor |C|/2 \rfloor$ edges of a). As in Lemma 3.1(iii) the total number of edges contained in all these subgraphs is $O(k\ell m) = O(k^3 n)$. Searching for length ℓ paths using color coding gives the following.

THEOREM 4.2. *A long cycle in an undirected graph can be found, if one exists, in expected time $2^{O(k)}n$ and worst-case time $2^{O(k)}n \log n$.*

Now we extend this result to find a long undirected cycle that contains a given vertex v (if one exists). The algorithm starts by deleting all vertices that are not biconnected with v. (Equivalently, it replaces the given graph by the subgraph H induced by the biconnected components containing v. Clearly the desired cycle lies within H.) Then it applies Theorem 4.2 to check if H contains a cycle C of length $> 2k$.

Case 1: C exists. Assume $v \notin C$, since in the opposite case we are done. C is contained in a single biconnected component B. This guarantees that B contains 2 vertex-disjoint paths from v to C, i.e., for each $i = 1, 2$ there is a vertex $x_i \in C$ and a vx_i-path P_i such that $P_1 \cap P_2 = \{v\}$ and $P_i \cap C = \{x_i\}$ for $i = 1, 2$. The desired cycle takes P_1 from v to x_1, then follows a subpath of C from x_1 to x_2 of length $\geq |C|/2 \geq k$, and then returns to v via P_2.

Case 2: C does not exist. In this case we seek a long cycle containing v with length $\leq 2k$. Such a cycle amounts to a vw-path of length ℓ, $k-1 \leq \ell \leq 2k-1$, for a neighbor w of v. Use color coding to search for paths from v of length $\ell = k-1, k, \ldots, 2k-1$. Since color coding finds all vertices w having a length ℓ vw-path, we find the desired path if it exists.

COROLLARY 4.1. *A long cycle in an undirected graph passing through a given vertex v can be found in time $O(m) + 2^{O(k)} n \log n$, if one exists.*

This corollary leads to a slight improvement for the problem of approximating the longest path in an undirected graph. Let ℓ be the length of a longest path. Björklund and Husfeldt [5] show how to find a path of length $\Omega((\log \ell / \log\log \ell)^2)$ in polynomial time. The bottleneck in this algorithm is finding a long cycle passing through a given vertex v. Using Corollary 4.1 for this task enables the algorithm to find a path of length $\Omega(\log^2 \ell / \log\log \ell)$ in polynomial time. This is proved by essentially the same analysis as [5]. We discuss similar algorithms for finding a long undirected cycle in a future paper.

Acknowledgments

The first author thanks Sudipto Guha and Sanjeev Khanna for pointing out the hardness results on the directed cycle problem. The authors also thank Thore Husfeldt for suggesting the application of our results to finding a long undirected path.

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading MA, 1974.

[2] N. Alon, R. Yuster and U. Zwick, *Color-coding*, J. ACM, 42 (1995), pp. 844–856.

[3] N. Alon, R. Yuster and U. Zwick, *Finding and counting given length cycles*, Algorithmica, 17 (1997), pp. 209–223.

[4] H.L. Bodlaender, *Minor tests with depth-first search*, J. Algorithms, 14 (1993), pp. 1–23.

[5] A. Björklund and T. Husfeldt, *Finding a path of superlogarithmic length*, Proc. 29th Int. Colloq. on Automata, Languages and Programming, Lecture Notes in Computer Sci. 2380, pp. 985–992, Springer Verlag, 2002.

[6] A. Björklund, T. Husfeldt and S. Khanna, *Approximating longest directed path*, Electronic Colloq. on Comp. Complexity, Rept. No. 32, 2003.

[7] M.R. Fellows and M.A. Langston, *Nonconstructive tools for proving polynomial-time decidability*, J. ACM, 35 (1988), pp. 727–739.

[8] M.R. Fellows and M.A. Langston, *On search, decision and the efficiency of polynomial-time algorithms*, Proc. 21st Annual ACM Symp. on Th. Comput., 1989, pp. 501–512.

[9] R. Impagliazzo and R. Paturi, *On the complexity of k-SAT*, J. Comp. Sys. Sci., 62 (2001), pp. 367–375.

[10] B. Monien, *How to find long paths efficiently*, Annals Disc. Math., 25 (1985), pp. 239–254.

[11] R.E. Tarjan, *Depth-first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.

[12] R. Tarjan, *Finding dominators in directed graphs*, SIAM J. Comput., 3 (1974), pp. 62–89.

[13] R. Yuster and U. Zwick, *Finding even cycles even faster*, SIAM J. Disc. Math., 10 (1997), pp. 209–222.

A The Tight Lowpoint Cycle Theorem

The goal is to show an upper bound of $(k-1)(k-2)$ in Theorem 2.1. We can assume $k \geq 4$. Several easy cases have long cycle of length $\leq 3k-6$. For space reasons we omit the arguments for these cases. We can assume the hypotheses of Lemma 2.9, $y \neq a$ and $C[a, f)$ does not contain a cross edge. We keep all the notation defined in that proof.

The proof of Lemma 2.9 easily implies that $|C[f, a]| \leq k-3$ makes $|C| \leq 3k-7$. So assume $|C[f, a]| > k-3$. Hence it makes sense to define h_s as the first high-water mark where

$$|C[f, h_s]| \geq k-2.$$

As noted in the proof of Lemma 2.9, $|C[h_i, h_{i+1}]| \leq k-2$ for every i. Hence

$$|C[f, h_s]| \leq (k-3) + (k-2) = 2k-5.$$

We can assume $h_s \neq a$ (otherwise a similar argument shows $|C| \leq 3k-6$).

We abbreviate T_{h_i} to T_i. Note that for any i, $C[f, h_i] \subseteq T_i$. This follows from Lemma 2.2(ii) with v the first vertex in $C[f, h_i] - T_i$ and $w = b = h_i$.

Now write

$$v = \text{the last vertex of } L_f \text{ belonging to } T_s.$$

(It is possible that L_f leaves T_s more than once. Also we might have $v = h_s = b$.)

The next result is useful for gluing C and C_{ef} together. In part (i) interpret paths as vertex sets.

LEMMA A.1. *(i) $C[f, h_s] \cap C_{ef}(v, e] = \emptyset$.*

(ii) $k \leq |C[f, h_s] + |C_{ef}[v, f]| \leq 3k-6-\delta$, where δ is an indicator variable for $v \neq f$.

Proof. (i) The definition of lowpoint cycle shows $C_{ef}(v, e] \subseteq L_f(v, r] \cup T[r, e]$, We have noted $C[f, h_s] \subseteq T_s$. By definition $L_f(v, r] \cap T_s = \emptyset$. We have $T[r, e] \cap T_s = \emptyset$ since $T_s \subseteq T_b$ and $e \notin T_b$.

(ii) For the first inequality note $|C[f, h_s]| \geq k-2$ and $C_{ef}[v, f]$ contains an edge leaving T_b and (e, f). The second inequality follows since $|C[f, h_s]| \leq 2k-5$ and $|C_{ef}[v, f]| \leq (k-1) - \delta$.

Now assume properties (i) and (ii) below. It is not hard to show that if either property fails, Lemma A.1 can be used to glue part of C to $C_{ef}[v,f]$ to get a long cycle of length $\le 3k-6$.

(i) $v \ne h_s$;
(ii) either $p(v) \ne h_s$, or
$v \in C[h_{s-1},h_s]$ and $|C[f,v],C_{ef}[v,f]| < k$.

(i) implies L_f leaves T_s. Write

$$\begin{array}{rcl} (v,w) & = & \text{the edge of } L_f \text{ leaving } T_s; \\ u & = & nca(v,w); \\ h_t & = & \text{the first high-water mark} \\ & & \text{that is an ancestor of } u. \end{array}$$

By definition u is a proper ancestor of h_s. It is possible that $u = h_t$. But first we must consider the possibility that h_t does not exist.

Lemma A.2. *If no high-water mark is an ancestor of u, i.e., $w \notin T_a$, then $|C| < (k-1)(k-2)$.*

Proof. Write $C = C[f,a], C[a,f]$. To analyze the length of the first path note that path $T[p(a),v],w$ is part of the lowpoint cycle C_{vw}. Hence $|T[a,h_s]| \le (k-1)-3 = k-4$. (We use (i) here.) As in Lemma 2.9 this implies $|C[h_s,a]| \le (k-4)(k-2)$. Thus

$$|C[f,a]| \le |C[f,h_s]| + |C[h_s,a]| \le (2k-5)+(k-4)(k-2).$$

Since $C[a,f)$ has no cross edge it is a shortcut of $T[a,e]$. The latter is part of C_{ef} by the hypothesis of the lemma. We get $|C[a,f]| \le (k-1)-2 = k-3$. We conclude $|C| \le 3k-8+(k-4)(k-2) < 3(k-2)+(k-4)(k-2) = (k-1)(k-2)$.

We can now assume h_t exists. We complete the proof by exhibiting a cycle D of the desired length, under the current assumptions. Consider the h_sf-walk

$$C[h_s,h_t], T[h_t,w], C_{ef}[w,f].$$

Define P to be the h_sf-path obtained by shortcutting the above walk. Define the closed walk $D = C[f,h_s], P$.

Lemma A.3. *D is a long cycle of length $\le (k-1)(k-2)$.*

Proof. To show D is simple it suffices to show that $C(f,h_s)$ is disjoint from the h_sf-walk that P shortcuts. $C(f,h_s)$ is disjoint from $T[h_t,w]$ since $C(f,h_s) \subseteq T_s$. $C(f,h_s)$ is disjoint from $C_{ef}[w,f]$ by Lemma A.1(i).

To show D is not too long we first estimate $|P|$. Note that $t > s$ so $t-1 \ge s$, and h_{t-1} properly descends from u. As in Lemma 2.9 we have these two inequalities:

$$|C[h_s,h_{t-1}]| \le (k-2)|T[h_{t-1},h_s]| \le (k-2)(|T[u,h_s]|-1),$$

$$\begin{aligned} |C[h_{t-1},h_t]| + |T[h_t,u]| &\le |C[h_{t-1},h_t]| + |T[h_t,h_{t-1}]| - 1 \\ &\le k-2. \end{aligned}$$

Since C_{vw} is not long, for some ancestor u' of u we have $1 + |L_w[w,u']| + |T[u',v]| \le k-1$. Lemma 2.4$(iv)$ shows $|L_w[w,u']| \ge |T[u,w]|/(k-2)$. Define

$$\delta = d(v) - d(h_s).$$

Thus $|T[u',v]| \ge |T[u,h_s]| + \delta$ and we get

$$|T[u,w]| + (k-2)(|T[u,h_s]| + \delta) \le (k-2)^2.$$

Combine the 3 displayed inequalities to bound the length of the first part of P:

$$\begin{aligned} &|C[h_s,h_t]| + |T[h_t,w]| \\ &\quad \le |C[h_s,h_{t-1}]| + |C[h_{t-1},h_t]| + |T[h_t,u]| + |T[u,w]| \\ &\quad \le (k-2)|T[u,h_s]| + |T[u,w]| \\ &\quad \le (k-2)^2 - \delta(k-2). \end{aligned}$$

Hence D has length

$$\begin{aligned} |D| &= |C[f,h_s]| + |P| \\ &\le |C[f,h_s]| + (k-2)^2 - \delta(k-2) + |C_{ef}[w,f]|. \end{aligned}$$

Now consider two cases. If $\delta > 1$ then use $|C[f,h_s]| \le 2k-5$ and $|C_{ef}[w,f]| \le k-2$ (since $f \ne w$) to get

$$\begin{aligned} |D| &\le 3k-7+(k-2)^2 - 2(k-2) \\ &= k-3+(k-2)^2 < (k-1)(k-2). \end{aligned}$$

In the opposite case, (i) implies $\delta = 1$ and (ii) implies $v \in C[h_{s-1},h_s]$ and $|C[f,v],C_{ef}[v,f]| < k$. This inequality, with $v \ne w$, implies $|C[f,h_{s-1}]| + |C_{ef}[w,f]| \le (k-1)-1 = k-2$. Hence

$$\begin{array}{rcl} |D| & \le & |C[f,h_{s-1}]| + |C[h_{s-1},h_s]| + (k-2)^2 \\ & & -(k-2) + |C_{ef}[w,f]| \\ & \le & (k-2)+(k-2)+(k-2)^2-(k-2) \\ & = & (k-1)(k-2). \end{array}$$

We conclude that D is not too long. To complete the proof we must verify that $D = C[f,h_s],P$ is not too short. By definition $|C[f,h_s]| \ge k-2$ so we need only show P contains ≥ 2 edges. Recall that P is an h_sf-path. It suffices to show (a) P enters f on the edge (e,f), and (b) $h_s \ne e$.

For (a) we need only show that f occurs just once in the paths that P shortcuts. This amounts to showing $f \notin T[h_t,w]$. By definition $f \in T_s$ and $w \notin T_s$, so we get $f \notin T[h_t,w]$ as desired. (b) holds because $h_s \in T_b$ (since $s < t$) and $e \notin T_b$ (since (e,f) enters T_b).